Survey of Existing Interactive Systems

Peter Bohlin, Johan Bos, Staffan Larsson,
Ian Lewin, Colin Matheson, David Milward

Distribution: PUBLIC

Task Oriented Instructional Dialogue
LE4-8314

Deliverable D1.3

February 1999

LE4-8314 TRINDI

Task Oriented Instructional Dialogue

Gothenburg University
Department of Linguistics

University of Edinburgh
Centre for Cognitive Science and Language Technology Group, Human Communication Research Centre

Universität des Saarlandes
Department of Computational Linguistics

SRI Cambridge

Xerox Research Centre Europe

For copies of reports, updates on project activities and other TRINDI-related information, contact:

The TRINDI Project Administrator
Department of Linguistics
Göteborg University
Box 200
S-405 30 Gothenburg, Sweden
trindi@ling.gu.se

Copies of reports and other material can also be accessed from the project's homepage, http://www.ling.gu.se/research/projects/trindi.

Contents

Chapter 1

Introduction

This report contains descriptions and evaluations of a variety of interactive systems which use or aid the use of natural language dialogue. The focus of our report is in particular on dialogue management techniques – how and to what extent these systems are sensitive to the current dialogue context, and how and to what extent they can use dialogue in furthering the user's objectives. The systems we examine exist now both in the commercial marketplace and in the laboratories of research organisations. The subjects of our study were chosen partly in order to represent the major threads of work current in natural language engineering and partly on account of their availability to different members of the Trindi project team. The subjects of our study include: Nuance's Dialog Builder, the CSLUrp dialog system toolkit, the Philips automated train timetable enquiry system, SRI International's Autoroute demonstrator system, the University of Rochester's Trains system and Verbmobil, a multi-site research project funded by the German State.

The report has two main sections. In the first, we survey the systems themselves and the role of dialogue management within them. In the second, we report the results of some investigation and evaluation of them.

1.1 The Nature of Dialogue Management

At the outset, one must emphasize that 'dialogue management' is a term that covers a great many different sorts of activity and phenomena.

Dialogue (or Conversation) Management, when applied to human-human interactions, might naturally be taken to cover the sorts of social and managerial skills manifested by a purposeful chair-person in a meeting. The chair-person guides, and sometimes explicitly directs, who can speak, when they can speak, for how long they can speak and even what they can speak about. Of course, these same issues arise in ordinary conversation outside the formal structures of meetings and, to greater and lesser degrees, people exhibit skill in managing them.

'Dialogue Management' as a term in the field of linguistic engineering includes these topics although they are not generally addressed to any great depth. The least ambitious, but commercially most advanced, of our subjects of study, Nuance's Dialog Builder application can be considered as a means for declaring who can speak when given one primitive for making utterances (from predefined strings or constructed from such), and another for recognizing them. Dialog Builder provides the means of defining the flow of control around these two basic operations. Paradigmatically, turns alternate and users speak when and only when prompted to do so. Most more ambitious dialogue systems do not attempt a great deal more in the social dimension of conversation management.

The term 'dialogue management' however encompasses a great deal more than this (even in Nuance's Dialog Builder). First, there is the the role of dialogue management and design in utterance understanding. For example, the fact that an utterance of 'Eindhoven' following the question 'Where do you want to go' is to be interpreted as meaning that the speaker wants to go to Eindhoven, is often taken to fall under the rubric of dialogue management. The reason is simply that 'Eindhoven' by itself is insufficient to license the interpretation. Only the context of the previous conversation can license it. So, an agent must have understood (or 'self-managed') the structure and meaning of the conversation so far in order to be able to understand the latest addition to it. This capability is quite independent of our earlier picture of conversational control and guidance. Another of our subjects of study illustrates this distinction particularly vividly. The VerbMobil system is designed to act as a conversational assistant in human-human conversations by supplying translations of contributions to non-native speakers. In order to do this, the system must have some understanding of the conversation so far but it has no interest in guiding or controlling the conversation itself. Indeed, the system does not even make contributions to the conversation except in an auxiliary way by helping others make contributions.

The making of contributions is a further distinct ability as witnessed by the fact that chairs of meetings may contribute to the discussions that they chair but it is not part of their function *as* chair to do so. Making a contribution requires one to be an agent of some sort, to have aims and objectives relating to the conversation and the means for devising and realizing those aims. The Trains system in our study is a particularly good example of a goal-directed rational-agent approach to making conversational contributions. In Trains, the user and system converse in order to establish a plan for transporting goods on a railway. The system can be viewed as continually attempting to infer the user's overall plan objectives from what the user actually says about what he wants. It can help satisfy these goals by reasoning over its knowledge of the transportation scenario. It makes its own contributions to the conversation based on its own specialist knowledge and on knowledge of general conversational maxims (e.g. that user requests should be acknowledged and addressed, and that false presuppositions in user utterances should be corrected). It is an interesting question how much general knowledge and action planning is in fact required for simpler domains and our other subjects of study (e.g. the Autoroute system and the Philips' train timetable enquiry system) employ computationally simpler mechanisms for determining their contributions to dialogue.

Chapter 2

The State of the Art in Dialogue Management

This chapter contains reviews of a variety of extant dialogue managers and dialogue design toolkits. It should be born in mind that the systems and components described strongly reflect the nature of the research and commercial projects within which they have been created and that these projects vary very greatly.

2.1 Dialogue Design Toolkits

2.1.1 Nuance Communications' Dialog Builder

Brief Description

Nuance Communications is a private company set up to exploit a particular instance of *speech recognition technology* (developed originally by SRI International's STAR LAB - Speech Technology and Research - in Menlo Park , California). Dialog Builder is a tool designed to help clients who purchase their speech recognition software to build complete applications involving speech output and speech input. The speech recognition software itself can be configured to return either a textual representation of what was uttered (or a list of the top n hypotheses) or a slot-value representation of the meaning of the utterance.

The key notion in Dialog Builder is *Dialog State* and, in its simplest form, the use of Dialog Builder consists of defining a set of such states, of which one is named the initial state, and a transition relation amongst them. Once the states and the transition relation are fully defined, one can execute the resulting spoken language language dialogue system incorporating both speech recognition and generation.

More formally, a dialogue state is defined by a triple

1. Name
2. Class
3. State Function

where the Name is simply a unique atomic identifier; the Class defines a place in an inheritance hierarchy permitting certain behaviours to be shared amongst states; and The State Function is arbitrary C-code, but it is required to define the *next* state for each execution path through it. (It is presumably called 'function' precisely because it must return the value of the next state.) Executing a dialogue therefore consists in executing the State Function of the initial state and, once it completes, executing the State Function of the next state as set in the execution-path through the just completed state.

In the abstract, this is simply a general programming architecture which embodies no particular theoretical picture of what a dialogue is. However, it is anticipated (in the Nuance documentation and tutorial material) that the simplest and paradigmatic sort of dialogue state is as follows

1. play a prompt to the caller
2. call the Nuance recognizer to recognize something and return a set of feature-value pairs
3. execute a case statement which, for each possible return, identifies the next state to execute

There are also a number of enhancements to this simple architecture.

First, there is the existence of the Class mechanism. Each dialogue state in Dialog Builder is associated with a Class (possibly null) which is a member of an inheritance hierarchy. Each class is associated with two hook-functions `entry` and `post-rec-fn`. Arbitrary C-code can be attached to these hooks. 'Entry' is called before anything else whenever a state in the given class is executed. 'Post-rec-fn' is called whenever the call to the built in recognizer function terminates. (There are special rules for handling cases of multiple inheritance).

The intention of the Class mechanism appears to be mainly to allow users to generalize both the setting of parameters for the speech recognition process before calls to the recognizer and the handling of the recognizer's return values (such as timeout, couldn't parse, hangup...) across dialogue states. For example, it is anticipated one will want to set the timeout parameter on speech recognition differently for different dialogue states, but it is too tedious to set it for *every* dialogue state. Similarly, one might want *always* to play "I couldn't hear you" whenever a timeout occurs, yet not code it for every dialogue state.

Secondly, Dialog Builder also permits a notion of "sub-dialogue" allowing one to execute one dialogue state recursively from within another dialogue state.

Thirdly, there is also a mechanism for re-executing a state having set a given flag.

Finally, there is a notion of "application-specific information" being an arbitrary (user-defined) association list of key-value pairs which is stored in a global structure. It is accessible (and updatable) in any dialogue state. So, for example, if you are currently within dialogue state s and the recognizer returns you a set of feature-value pairs (corresponding to the semantics of an utterance), then you will need to copy them into a piece of global state if you want to access them again in any later dialogue state.

Discussion

We will discuss Dialog Builder under three separate headings: the possible influence of dialogue on the interpretation of subsequent utterances, the mechanisms available for deciding and making a contribution to the dialogue and the 'social' features of dialogue management, including such things as turn-management.

How much in-built support is there in Dialog Builder for making the interpretation of user utterances dialogue sensitive? The in-built speech recognition function (from input audio to output text or slot-value pairs) is only context sensitive in one way. One can specify the grammar that the speech recognizer will use. Therefore one way to make interpretation context sensitive is to specify different grammars for different stages of the dialogue, loading them when they become appropriate. The facility is provided in order to be able to constrain the recognition process but it seems it can also be used to constrain the slot-values interpretation which is specified inside the grammar. More concretely, in a banking application, if one wants to make "Savings" mean $source - account = savings$ after the question "Which account do you want to transfer from?" and mean $destination - account = savings$ after the question "Which account do you want to transfer to?" then you could accomplish this by loading different grammars at runtime after having output the relevant prompts. Alternatively, and certainly more attractive, at least in this instance, one can use one grammar that *underspecifies* the meaning of the input utterance and leave it to dialogue management code to fill the specification. In practice, since dialogue states are intended to correspond fairly closely with possible prompts to the user, one is liable to encode one dialogue state for one question and a different dialogue state for the other question. Then one can easily "hard-code" within the State Functions of these states that $account = savings$ maps to $source - account = savings$ within the "obtain source account" state and to $destination - account = savings$ in the other. The remaining option is to use a piece of the global information structure ('application-specific information') in order to encode the required dependency.

The role of conversational knowledge and state in influencing the making of utterances in a Dialog Builder application is rather less clear. The primary problem is that the decision about the next dialogue state to enter into must be taken *within* the current dialogue state but dialogue states are not primarily to be thought of as *information states*. They appear to

resemble more what a linguist would call a *sub-dialogue* or *game* because each paradigmatically consist of one simple interchange between system and user which extracts a certain piece of information from the user. For example, the get-source-account state will be a state that queries for an account name and updates the global variable that stores the name of a source account. The state can be entered at various different stages in a dialogue and also re-entered later on if necessary. On this picture, however, it is odd to require that executing a dialogue state must result in being able to name the next dialogue state to enter, as is the case in Dialog Builder. In practise this means that the dialogue states that a designer codes for one application cannot easily be transferred to another application. Even if one designs another banking application and wishes to obtain the name of the source account, one cannot easily re-use the dialogue state which implements this sub-dialogue because the existing state names states in the original application.

One would naturally expect, if Dialog Builder's dialogue states resembled sub-dialogues , that the decision about the next state to enter (and hence the next prompt to issue) would be made *after* completing the sub-dialogue. Unsurprisingly, in illustrative examples in the accompanying documentation, the decision is in fact always the last action that takes place within a dialog state. Nevertheless, if we are considering the influence of dialogue information state on the next move decision, we are still left with the position that the decision is at least open to influence by information which is purely local to the current state. This is presumably motivated primarily by the need for natural language equivalents of menu-traversal – "Do you want to make a transfer, query your balance or pay a bill" whose answer immediately leads one to another portion of the dialogue space and has no other influence on the rest of the dialogue. If one wanted to encode a dependency such as 'I will ask for the source account next so long as I do not already know it', then one would again need to use the globally available 'application-specific' information structure.

According to the Nuance tutorial literature, the paradigmatic dialogue state consists of the system issuing a prompt and the user replying. A complete dialogue consists of a sequence of such states. In social terms, this means that the system is generally always retaining the initiative in the dialogue and that there is strict turn-taking. Indeed, the built-in function to call the speech recognizer plays any prompts declared so far and then waits for user input. Dialog Builder is designed to be maximally flexible so it is possible, for example, to play prompts outside of calls to the recognizer, or to recognize without prompts, or, it appears, to call the recognizer as many times as you like within one dialogue state. The paradigmatic picture may appear very constraining but it can be motivated by the observation that guaranteeing high levels of accuracy in speech recognition itself remains a primary objective for any system involving speech. In commercial applications, users still have to be guided very carefully into making utterances that the speech recognition software has a good chance of being able to understand. Poor quality speech recognition will result in poor quality applications, no matter flexible the dialogue manager.

2.1.2 The CSLU Speech Toolkit

Brief description

The CSLU Speech Toolkit, developed by the Center for Spoken Language Understanding at Oregon Graduate Institute of Science and Technology is designed to be a comprehensive environment for research and development in spoken language systems. [1] The toolkit integrates a set of core technologies including speech recognition, speech synthesis, facial animation and speaker recognition. It also features authoring and analysis tools enabling quick and easy development of desktop and telephone-based speech applications.

The CSLU rapid prototyper (CSLUrp) part of the toolkit is intended to offer a user-friendly authoring system for spoken dialogue systems, incorporating a drag-and-drop graphical interface. CSLUrp provides an interface to neural network-based speech recognisers, including dedicated number and letter recognisers, a speech synthesiser, and the library of C routines that operate these and link them together. In its simplest form, the prototyper works on a finite state model of dialogue, where each state optionally includes an audio prompt (synthesised or recorded) and a speech recogniser. In order to build a simple dialogue system, users drag recogniser objects (states) onto the canvas, specify the recognition vocabulary or grammar for each state, and link these states together in the required order. If moving to a state is to be dependent on the output of the recogniser, this can be easily achieved by specifying a different vocabulary for each output connection.

CSLUrp includes a default speech recogniser which the user can manipulate with minimal effort, as well as offering the flexibility of specialised or user-constructed recognisers. The default recogniser is speaker- and vocabulary-independent, and is constrained by the user at each individual state by specifying which words or combinations of words form legitimate strings at that point in the dialogue. The recogniser automatically performs word-spotting for any of the listed vocabulary listed, although the user can include grammars in the form of basic regular expressions if required.

Speech output is provided by recorded prompts or by a synthesised text-to-speech (TTS) system. Prompts can be recorded by the developer using any application which writes .wav files – a library routine for recording prompts in CSLUrp can be used, or a customised routine can be created using CSLUrp. The TTS system which comes with the CSLU Toolkit is Festival, a speech synthesiser developed at the University of Edinburgh. Once installed, use of the synthesiser from within CSLUrp is very straightforward – all text typed into the prompt box on the recogniser object is by default rendered in TTS.

One of the key features of CSLUrp is that the user can create a system without doing any programming. Objects can simply be moved around on the graphical interface, connected, and associated with input vocabulary and prompts. At any point the graphical representation can be compiled into a program and run. However, if the user does wish to create more complicated systems, or make the next state dependent on some calculation other than the

[1] This description is taken, largely verbatim, from [Davies(1997)]

output of the state's recogniser, the prototyper supports the Tcl scripting language. Tcl code can either be typed into the action boxes on the recogniser object, or can be created in any text editor and imported. The ability to use arbitrary code to move from dialogue state to dialogue state does, of course, move the process out of the finite state class.

Discussion

As with Nuance's Dialog Builder, from the point of view of the TRINDI project there is no dialogue manager in CSLUrp as such – the program is designed to assist in the development of dialogue managers, rather than constituting one itself.

The general principles behind the CSLUrp toolkit closely resemble those of Dialog Builder and so many of the comments applied to Dialog Builder apply also to CSLUrp. The basic method for constructing a dialogue using CSLUrp is to select and join graphical objects to form a finite state automaton or 'flow chart' of the dialogue structure. Dialog Builder lacks the graphical interface but also contains some additional structuring in the form of state classes.

The CSLUrp toolkit provides no mechanism for interpreting user utterances, only recognizing them. The only supported sensitivity is to the current recognition grammar and vocabulary which may vary from state to state. Consequently, there is no in-built support for making interpretations dialogue sensitive. One can code extra sensitivity by writing suitable code in the Tcl scripting language of course. Interestingly, Nuance's Dialog Builder does provide a degree of extra support here through it's support for an association list of global information together with a primitive operation of priority union on such slot-value structures.

The role of dialogue state in influencing the making of utterances is as unclear in the CSLUrp toolkit as it was for Dialog Builder. The nodes in the flow chart *can* be seen as information states (encompassing all one needs to know in order to progress a dialogue) or in the more limited sense of dialogue game states (encompassing only the general structure of the dialogue rather than what the participants know or have agreed). Which view is the more appropriate one depends to a great deal on how much information and calculation is to be carried out within the Tcl code that one can attach to dialogue states. If there is none, so that progressing through the whole dialogue can be viewed as essentially traversing a menu-structure, then the states resemble information states. However, the more that decision-making becomes independent of the network diagram itself, then the less that the states in that network appear to be information states. It is also worth remarking that the more Tcl code one attaches to the networks the more likely it is that the advantages of using CSLUrp become dissipated. The problems of understanding, manipulating and importing code are likely to outweigh the benefits of the graphical display – which may, if due care is not taken, increasingly misrepresent the underlying dialogue structure.

The social management functions of the CSLUrp toolkit resemble those of Nuance's Dialog Builder very closely. States are intended to consist of playing a prompt (a recorded audio file, or an external call to a text-to-speech synthesizer) and then calling the supplied speech

recognizer. The somewhat constraining nature of this picture is again justified by the primary consideration of maintaining high levels of speech recognition accuracy.

2.2 Dialogue Management Systems

2.2.1 Philips

Brief Description

The Philips Train Timetable System is the result of a demonstrator research project which is focussed on a particular application and is not intended directly for sale to customers (except possibly, the German Railway operators themselves). Nevertheless, the system is also designed not to be wholly dependent on the domain of German train timetables and the system has been ported elsewhere (notably as a train timetable system for the Netherlands). The Dialog Control Module is intended to be generally applicable to the domain of *automatic inquiry systems.* Here we describe the system described in [Aust and Oerder(1995), Aust *et al.*(1995)].

The Module is implemented through a special purpose dialogue programming language and a dialogue interpreter. A program in the dialogue programming language consists of a number of sections which are best understood through an understanding of how the dialogue interpreter exploits them. The sections are

- a *General Framework* section
- a *grammar rules* section
- a *slot definitions* section
- an *Extension Questions* section

The *General Framework* section specifies the highest level of dialogue structure. It appears to be essentially a procedurally defined object which in the train timetable application, executes the following

1. an initial greeting
2. an initial question
3. conduct dialogue extending the answer into a possible database query
4. make query and present results
5. query 'do you want anything else?'

- if "no", terminate
- if "yes", go back to 2

The sub-dialogue which constructs a possible database query is the most interesting phase of processing. Indeed all the other sections of the dialogue programming language (grammar rules, slot definitions and extension questions) are parameters to this process. The database query to be constructed consists of a list of filled slots (such as destination, origin, time...) and each answer that the user gives is expected to help fill in some of these slots. When some predefined subset of the possible slots have been filled, a database query can be executed.

In order to fill in these slots, the sub-dialogue cycles through the following process

1. If there a disambiguation question applicable to a slot, ask it
2. Else if there is an extension question, ask it
3. Else finish sub-dialogue

The point of disambiguation questions is to clarify how a previous answer is intended to fill in a slot, as a result of detected ambiguities, vagueness or apparent contradiction. There is a set of rules (format unspecified) governing the combination of slot-values. For example, if one slot is filled with both "five o'clock" and "in the afternoon", then these values can be combined to generate "five p m". If the slot were only filled with "five o'clock", then a disambiguation question would be required. One would also be required if it were filled with both "five o'clock" and "nine o'clock" (perhaps owing to a mis-recognition). These sorts of question are always asked first so that a coherent sequence of questions is maintained.

The point of extension questions is to fill in slots whose values are required but as yet unknown. Each possible question is associated with a pre-condition. Typically, one might have a pre-condition of "The destination is unknown" associated with the question "Where do you want to go?" Presumably the pre-conditions also have an ordering imposed upon them in some way since many pre-conditions may be true at any one time.

Finally, any question to be asked of the user is subject to *verification processing* in which slot-values which the user has supplied but not confirmed in previous answers are inserted into the next question for *indirect confirmation.* For example, if the system thinks the user has said he is going to Munich then the next question 'When do you want to go?' is transformed by verification processing into 'When do you want to go to Munich?' Interestingly, this strategy of indirect confirmation was chosen because of high recognition failures on direct confirmation questions such as 'So you want to go to Munich'. Verification processing is carried out using some sort of pattern based processing but the format of the patterns is not discussed and it is unclear how easy it might be to reconfigure this processing stage.

Discussion

The heart of the Philips' system is based upon the idea of *slot-filling*, this being the result of a process that begins with speech recognition and the primary input to another level of dialogue processing. The key supported elements in Philips' dialogue description language are sections for combining slots (or issuing disambiguation questions) and rules for deciding which slot to ask about next. The set of slots that have already been filled constitute the dialogue *information state* on the basis of which the next question is asked. That is, the next move to make is decided upon by examining which slots have already been filled and what they have been filled with (so that disambiguation questions can be asked first). The Philips' system incorporates a certain amount of reasoning over these information states. For example, it needs to be able to reason that "four o'clock" and "in the afternoon" can both fill the same time-slot but that "Hamburg" and "Berlin" cannot both fill a destination slot. It also performs some elementary reasoning about what it does not know and decides what to find out about. In Nuance's Dialog Builder, the dialog designer has to provide code determining what the next dialogue state should for any execution path through the current state. The Philips' system can be viewed as providing a certain amount of automated support for this decision.

It appears, from the published descriptions, that the interpretation process is generally *not* dialogue sensitive. Each user utterance passes through stages of speech recognition and speech understanding but neither stage appears to take a dialogue state as a parameter. For instance, it would appear that if the system asks a question such as "Where do you want to go" and the user replies with "I want to arrive at four p m", then this is treated just as if the system had asked "When do you want to go?". The system is entirely unaware that the original question was not *answered.* It is an interesting question what the cost-benefit analysis of such a policy is. If the user ignores the question actually put either because he has more important information to impart or because he misrecognizes what the system says, then the policy has the advantage of not throwing away useful information provided by the user. This is only true of course if the system *can* determine reliably the meaning of what the user actually says independently of the context that the system thinks is current. There is a danger of the policy permitting further and particularly complex misunderstandings which are difficult to recover from.

[Aust *et al.*(1995)] contains two sentences which refer to modification of the speech understanding component 'in order to account for the dialogue state that led to the question' and modification of 'answer processing' during a verification. Unfortunately, there is no further elaboration on what such modifications amount to. It appears most likely that the Philips system contains hooks which permit a certain amount of dialogue sensitivity to be added to the basic system in cases where it is deemed essential, but such sensitivity is not a built-in feature of the architecture itself. It is worth noting that, in general, verifying a user utterance may often result in context-sensitivity. For example,

(2.1) So you want to go to Hamburg?

(2.2) No, Homburg.

The utterance 'No, Homburg' does not invoke the concept of a destination, which must be inferred from the verifying question. The prevalence of this type of interaction in current, live, fielded systems (where commercial speech recognition may still be highly errorful) will require a certain amount of context-sensitivity.

The Philips' system employs a strict turn-taking mechanism and, it appears, retains the initiative at all times.

2.2.2 SRI-Autoroute

Brief Description

The SRI-Autoroute system is a demonstrator system which provides a natural language interface to the popular PC-based Autoroute route planning package. The system resulted from adaptation of the existing reconfigurable AURIX speech recognition system [Russell *et al.*(1990)], a reconfiguration of SRI's Core Language Engine [Alshawi(1992)], a generic parser and meaning generator for utterances and text, and the addition of a new dialogue manager designed to exploit the notion of conversational game, and also designed to promote reconfigurability.

The dialogue manager contains two distinct levels of management. First there is a fairly simple implementation of a "rational agent" who has a set of beliefs about the world and actions that it can undertake and a goal. The goal is simply to present a route-plan to the user. An action consists of a precondition (something that is required to hold before the action may take place), the name of an executable object (the action itself) and a postcondition (something that will be true if the action does take place successfully). In order to decide what to do, the agent draws up an initial plan consisting of a sequence of actions which it believes will lead it into a state where its goal has been satisfied. Most of the actions it plans are "conversational games", these being small rule-governed interactions between system and user. The details of individual moves within games are not planned, just as one might plan to play a game of chess without planning the moves one might make within the game. After each game completes, the dialogue manager assesses whether the game succeeded or not and either carries on with the rest of its plan or makes up another plan to replace the original one. The dialogue manager may also choose to skip over an action in a plan if its postcondition has already become true by the time it is due to be executed.

The Conversational Games are defined by recursive transition networks whose arcs are labelled with the names of conversational moves. For example, a very simple "questioning game" might consist of a linear network labelled with "question" followed by "answer" followed by "acknowledgment". Games are executed by the dialogue manager by traversing the networks that define them. Traversal is a hill-climbing search with back-tracking. That is, whenever a choice is required, the set of possible next moves is sorted according to an evaluation function and the highest scoring move is chosen. There is a predefined set of scoring functions and associated weights. For example, one scoring function predicts the expected length of a game following a given move. Another calculates expected accuracy of understanding. If a search

path runs into a dead-end, the system backtracks to the last open choice point and picks the next highest scoring move. All possible analyses are therefore tried eventually. The system may also backtrack into re-analysis of its own utterances.

On the user's turn, the CLE is called to determine the syntax and semantics of the utterance. A move recognition process uses these plus the current game state (current node and propositions under discussion) to determine the possible move analyses. Then, the preference mechanism is called to choose between them.

On the system's turn, the set of next possible moves is calculated from the game definitions and the preference mechanism is called to choose between them. Then, a move realization process determines an utterance meaning for the chosen move and the CLE is again called to generate a string from the meaning. (Most calls are cached in advance to save processing time but some meanings such as the contents of checking moves cannot be predicted).

Discussion

We consider first the influence of dialogue context on the interpretation of utterances.

The inputs to the Autoroute dialogue manager are the outputs of the Core Language Engine, which itself contains a large amount of context dependent processing machinery, for example for handling pronoun resolution and ellipsis. Consequently, both the Core Language Engine and the dialogue manager must maintain their own dialogue contexts (and they need to be kept in step with each other – in practice, the Core Language Engine tags all context-dependent items with an utterance identifier and the dialogue manager tells the Core Language Engine what the dialogue structure over utterances is). It also means there is the possibility of informational duplication between the Core Language Engine and the dialogue manager, since they are both keeping copies of a dialogue context. The justification ([Lewin and Pulman(1995)]) for keeping the processing within the Core Language Engine is that, for example, in order to fill in an ellipsis, appeal should be made to the structure of syntax or semantic representations, which is a level of detail that dialogue managers are generally ignorant of. Furthermore, by treating these processes linguistically the overall resulting system is more easily reconfigurable. If one considers a new domain (but the input language is one which the Core Language Engine already covers), then one will not need to reconfigure any part of the dialogue manager in order to handle elliptical utterances.

There is also a possible dependency of interpretation upon context within the Autoroute dialogue manager itself. The Core Language Engine is intended to deliver the 'propositional content' of input utterances and the dialogue manager is intended to further classify the utterances, based on their content and the dialogue so far, as conversational *moves*. The notion of a move is the notion of *performing an action* in virtue of saying something and bears comparison with the notion of speech-act. However, such move ambiguities (such as 'You will travel to a foreign land' being misconstrued as a command rather than a prediction) hardly if ever occur in the Autoroute scenario itself.

The two-level nature of the Autoroute dialogue manager gives rise to two different decision making processes when considering what action to undertake next. If the dialogue manager is executing a conversational game, then it examines its next possible moves according to the transition networks and chooses amongst them. In practice, these decisions amount to fairly simple ones such as "Shall I acknowledge that utterance?" and "Shall I verify I heard that utterance correctly?" These decisions are taken by a simple expected utility function. It is not difficult to imagine that, in the general case, a complex reasoning engine might be needed. The second level of decision making is which conversational game to play next. In Autoroute, given a current plan, this decision amounts to:

1. if the last game failed, re-plan

2. if the next game's post-condition is true, skip it

3. else perform next game in plan

The main reason for this structure is simply to avoid the computational overhead of using general purpose planning many times within what is actually a successful conversation.

The Autoroute dialogue manager enforces strict turn-taking in two places. At the top-level, the "rational agent" is continually making plans and executing them and these plans include initiating conversational exchanges. Consequently the system clearly retains all initiative at this level. Indeed, the system will not even be listening to its partner if it has not decided that it is playing a conversational game and that it is the partner's turn to say something. Consequently, a user cannot interrupt or break in. Within a conversational game, turn-taking is also enforced for similar reasons as each node in the transition network is labelled with the name of the person whose turn it is. In this way, the system always knows whether it should be listening or talking.

Unlike the Philip's system, the Autoroute dialogue manager does know that it has asked a particular question and seeks to establish the answer. As already noted, this poses a potential danger that, should the user provide useful information but not actually answer the question (for whatever reason) or indeed answer the question but also supply extra information, then the useful information will be thrown away. In the current Autoroute system, part of this danger is averted through the definition of "answer". An utterance answers a question if its content contains elements that answer it. Further, the dialogue operation associated with an answer is to update the context with the whole meaning of the utterance. Consequently, useful extra information is not thrown away. Useful utterances which do not answer a question require a more complex treatment. In principle, if the question game permits a nested assertion game to follow a question, then the user's utterance can be analyzed and incorporated into the current game.

2.2.3 Trains

Brief Description

The basic scenario for the Trains system is route-planning. The system aims to provide help to a user whose task is to move a number of trains to specific destinations in the most efficient manner. A map is presented of the North East of the US and South East of Canada, with train lines and cities marked. Then, a goal is presented to the user, for example,

SYS: `The engines are currently at indianapolis, lexington, and toronto. Your goal is to get them to montreal, pittsburgh, and buffalo.`

The user then conducts a spoken dialogue (or typed, if speech recognition is not available) with the system in order to achieve the goal. At the end of a session the system will also assess the difficulty of the chosen task, estimate how many interactions should be necessary to complete it, and thereby give a "score" to the user, for example

SYS: `You took 1 minutes, and made 5 interactions with the system. I expected you to take 4 minutes, and make 14 interactions with the system, for this level of problem. I rated this problem as fair, which gives you a score of 3, with an average score of 100, and includes a bonus of 3. A hopeless score!`

The Trains system has existed in a number of different forms (see, for example, [Ferguson *et al.*(1996), Allen *et al.*(1996b), Allen *et al.*(1994), Allen *et al.*(1996a)]). The particular incarnation we consider here is the "robust" version Trains-95 ([Allen *et al.*(1996a)]), partly because this version is also available to Trindi project members for empirical evaluation.

The Trains system embodies a strong plan-based approach, by which is meant that the system is continually attempting to infer the user's overall plan objectives from what is actually said. The system is not designed to compute only the "strict and literal interpretation" of what a user says, but is designed to attempt to infer the user's intentions behind what he says. In this way, it is hoped the system can be more co-operative by being sensitive to the user's objectives rather than simply what is uttered. In general, plan recognition is a hard problem and computationally expensive. Consequently, the "robust" Trains-95 system substitutes domain-specific reasoning techniques for general plan-based reasoning whenever possible.

The Trains system also includes another planning element – the train route-planning component itself. The system helps the user in forming a route plan partly by performing some route-planning itself. This component is deliberately fairly weak so that further interaction between the user and the system is required. The system cannot plan route sections longer than four hops and when there are plan alternatives, the system simply picks one at random.

The Trains dialogue manager takes for input a set of speech acts, these being analyses of the

input utterance which include a component identifying illocutionary force. For example, a simple utterance such as "okay" may be classified as a confirmation. This tag shows what the user may be *doing* with his utterance as well as whether the utterance is grammatical and what semantic content it may have. The illocutionary force tags applied by the parser are provisional ones only. Discourse processing consists of several stages. First, reference resolution identifies the referents of linguistic expressions (for example, that "the train at Atlanta" refers to train number 4672) and may also update the illocutionary force associated with a speech act. Secondly, inputs may be further interpreted by a "verbal-reasoning" component. This is essentially an ordered set of pattern-matching rules over speech-acts and dialogue states and is designed to provide a degree of robustness in the face of ill-formed or fragmentary input. The problem-solver is the component which attempts to update the system's idea of the the current plan given the new inputs. It is also (in Trains-95) a prioritized set of pattern matching rules.

Trains maintains a dialogue state which is a stack of elements, each of which identifies

- the domain or discourse goal motivating the current discourse segment
- the object focus and history list
- information on the status of problem solving (e.g. has the current goal been achieved?)

Discussion

The Trains system incorporates a number of ways in which the interpretation of utterances can be made sensitive to dialogue context. The simplest case is that of reference resolution. The dialogue state contains an object focus and a history list (resembling the attentional state structures described in [Grosz and Sidner(1986)]) which constrain the interpretation of linguistic expressions. Furthermore, the verbal reasoner and the problem solver are both explicitly represented as patterns to be matched against both the input and the current dialogue state. Consequently, these two stages of processing can also be made dialogue sensitive in a very straightforward way. Naturally, the issue arises of how easy it might be to write such rules and how domain-independent any given set of rules might be. The importance of the issue is openly acknowledged in [Allen *et al.*(1996a)].

The general domain-reasoning component of Trains introduces another important sensitivity into utterance interpretation. It is possible for "early" linguistically-based processing to interpret an input one way but for domain-reasoning to conclude that that interpretation is not likely on the basis of the current problem solving state (regardless of where one is in the dialogue and how one got there). For example, suppose an explicit user rejection "No, let's take the train from Detroit to Washington" is mis-interpreted by "Now let's take the train from Detroit to Washington". The latter interpretation suggests an extension to the current plan. The problem solver may however detect an error in this plan. Perhaps there is no train at Detroit currently. The user's rejection was intended to supplant the current plan

with one where there is an engine at Detroit. The problem-solver can reject the inconsistent interpretation and ask for a new one. Whether one can be found is another matter.

The action that the system decides to take next depends on the goal stack maintained as part of its dialogue state. Consequently, this part of the system can also be made sensitive to the current dialogue state in interesting ways. The details depend on how particular items are added to the agenda and how they are taken off. For example, in the version of Trains-95 described in [Allen *et al.*(1996a)], an "okay" interpreted as a confirmation can have the result of popping an outstanding yes-no question off the top of the goal stack. If however there is nothing suitable on the top of the stack to take off, then the utterance is simply ignored rather than prompting a query or clarification. This is presumably because the system is designed with maximum "robustness" in mind and it is probably safer to assume that the current interpretation may be wrong but is, in any case, insignificant overall than to assume that the correct interpretation of the "okay" must be gained before anything else useful can be done.

In terms of social management, the Trains system is interesting in that, although it certainly employs a strict turn-taking regime, the nature of the domain in which it operates means that the user can be considered generally to hold the initiative. Dialogues tend to consist of the user making suggestions and asking questions and the system responding to them. This is in marked contrast to the Autoroute and Philips' systems, for example, where the system generally asks the questions and the interpretative task is one of making sense of the answers. The system also can take some sort of initiative upon itself. For example, if the user asks for a route from A to B via C and the system knows that this route is poor then it may choose to issue a warning to the user: "The terminal at city Scranton is delaying traffic due to localized heavy winds". However, it is also possible to interpret such warnings as replies to the user's request which are rejections in the form of warnings.

2.2.4 Verbmobil

Brief Description

Verbmobil is a large distributed project (funded by the German Federal Ministry of Education, Science, Research and Technology (BMBF)) for speech-to-speech translation of negotiation dialogues, between three different languages: English, German, and Japanese. The overall project time is divided in two phases. Phase I finished at the end of 1996 resulting in the *research prototype* [Bub and Schwinn(1996), Bub *et al.*(1997)]. Phase II is still underway, aiming at a prototype that improves both in linguistic and domain coverage.

In the first phase of the Verbmobil project the scenario was restricted to appointment scheduling dialogues — a task of negotiation, where two (or more) participants try to agree on a date to meet at a certain location. This scenario was extended to travel planning in the second phase of Verbmobil, putting an extra demand on the system to deal with information exchange dialogues as well.

The task of the dialogue module in Verbmobil is monitoring the conversation between the participants resulting in a representation of the dialogue history and predictions of the dialogue moves—information that is provided on request to the ambiguity resolution and translation modules. Facing a lack of controlling the dialogue, the dialogue module must be able to follow the dialogue in any direction covered by the scenario. At this point Verbmobil differs considerably from other interactive dialogue systems discussed in this report, as it needs not to decide for a next move in the dialogue.

Central to the dialogue module is the dialogue history (Verbmobil's "information state"), the database with context information on which inferences are drawn and dialogue acts are predicted. It has three parts [Alexandersson *et al.*(1997)]:

- **dialogue sequence memory:** the sequential order of turns and utterances related to speaker, dialogue act, and move predictions (within Verbmobil dialogue processing the terms *turns* and *utterances* are standard to refer to parts of the conversation. A dialogue participant contributes to the dialogue using *utterances*, and the sequence of utterances uttered makes up the *turn.*);
- **intentional structure:** tree like structure (leaves are dialogue acts, nodes are turns of dialogue phases) representing the "dynamics" of the dialogue;
- **thematic structure:** the propositional content (restricted to time expressions) related to the attitude of the different dialogue participants (propose, reject, or accept).

Inference mechanisms allow integration of data in representations of different aspects of the dialogue. There are three kinds of inferences. Firstly, there is a *plan recognizer* which determines the phase of the dialogue (in Verbmobil: opening, negotiation, or closing). Secondly, there is a set of inference rules working on the thematic structure, that deal with the analysis of deictic time expressions (e.g., "next week"), the exclusion of implausible time expressions (for instance "the 30th of February"), or the implicit rejection of proposals if participants propose different dates for a meeting. Finally, the dialogue sequence memory serves dialogue act prediction. Predictions are trained on the annotated Verbmobil-corpus and are computed using the conditional frequencies of dialogue act sequences to compute the probability for following dialogue acts, information that is used to constrain disambiguation [Alexandersson *et al.*(1997)].

Dialogue acts ("moves" in the Trindi terminology) express the primary communicative intention of an utterance (or a dialogue segment). They are defined upon a hierarchical tree structure, allowing more fine grained classification of moves if possible. On the top level we find CONTROL-DIALOGUE (social interaction such as opening or closing the conversation), MANAGE-TASK (initializing, deferring, or closing the task), and PROMOTE-TASK (anything relevant to the task itself). Each of these branches can be further refined, resulting into a set of more than 30 different dialogue acts [Alexandersson *et al.*(1998)]. Naturally, deeper down in the hierarchy, the finer grained dialogues acts are highly domain specific, tuned down on Verbmobil's task of appointment scheduling.

Discussion

Dialogue management in Verbmobil has great impact on the interpretation of utterances. Apart from the role dialogue acts play within the dialogue memory (as outlined above), they contribute to various aspects within Verbmobil, ranging from disambiguating the user's input, via template choice decision, to clarification dialogues.

A simple example illustrates how dialogue acts help in the disambiguation process while translating natural language expressions. The German verb *wiederholen*, for example, translates in an opening or negotiation phase to the English *to repeat*, whereas in a closing phase of the dialogue it is likely to be translated as *to summarize* [Alexandersson *et al.*(1998)].

Further, dialogue acts steer the selection of templates for generating target language expressions. Verbmobil consists of various competitive processing streams, divided into *deep processing* (roughly, those analyses that make heavy use of linguistic knowledge) and *shallow processing* (those components that exploit keyword spotting techniques and statistical algorithms for translation). It is the latter line of modules within Verbmobil that use dialogue acts for generating the surface target translation.

Clarifications dialogues in the Verbmobil system allow the user to interact with the system. (An example is the system asking the user to choose among different analysis, when e.g. ambiguous time expressions were used in the utterance.) Currently this application hasn't been fully explored, and little or no information on dialogue moves are used in the implementation. The second phase of Verbmobil will probably place heavier demand on the use of dialogue acts in clarification dialogues, where a summarization functionality will be supported. The idea here is to use dialogue acts to determine the directions taken in a dialogue, and from this the core of the conversation is deduced and taken as a base representation for a summary [Alexandersson *et al.*(1998)].

Chapter 3

Dialogue Manager Evaluations

3.1 How can Dialogue Managers be evaluated?

How properly to evaluate dialogue systems and dialogue managers as components within them is a notoriously difficult topic. A good review can be found in the Eagles (Expert Advisory Group on Language Engineering Standards - EC-DGX111,LRE,LRE-61-100) "Standards" handbook [Gibbon *et al.*(1997)].

In general, assessments are generally divided into Black Box and Glass Box methods. The assessment of inputs and outputs without any attempt to "look inside" to see what causes them is called Black Box assessment. Possible metrics for this sort of analysis include, amongst others, task-completion rate, task success rate, average length of a dialogue (in terms of utterances, turns or time) and user satisfaction.

The evaluation of the contribution of components to overall functioning of a complete system is called Glass Box assessment. For example, one can use word-error rates to evaluate the performance of a speech recognition sub-system. This is because speech recognizers have, currently, a clearly defined task, namely the production of word strings. (Note, it immediately becomes more complex to assess a recognizer if generates something else, such as slots and their values, as the Nuance speech recognizer can be configured to do). Unfortunately, it is not straightforward to apply this idea to dialogue managers, since it is not clear what the core functionality of a dialogue manager ought to be. The Eagles Handbook remarks that there is no stable categorisation of the basic units used in dialogue systems. This makes cross-system comparisons very difficult. The issue is further complicated by the fact that, in general, only system developers have access to dialogue managers as a separate component (if it is indeed separable) of a complete dialogue system. Without this access, an evaluation can proceed only on the basis of the input-output system behaviour and the contribution of dialogue management to this behaviour may be highly uncertain.

The investigations which follow in this chapter are attempts to examine whether certain

characteristic behaviours which one would expect in a system deploying certain dialogue management principles can be reliably manifested by that system. To carry out such an examination, we have constructed a "Tick list" of yes-no questions which examiners attempt to answer by carrying out dialogues with each system under scrutiny. These examinations are designed both to tell us something about the systems and also to help us develop the "Tick-list" itself. The "Tick-list" is not a fixed point. We expect to develop the Tick-list further in consultation with the Trindi International Consultative User Group and place it on the World Wide Web.

We have carried out two sorts of examination of those systems to which Trindi members have access at the time of this report. First, based on published accounts of the systems, we have applied the Tick-list criteria. We have called these 'examinations from the theoretical perspective'. Secondly, we have applied the Tick-list empirically by conducting real dialogues with the systems.

Undoubtedly, our examinations do not bear the precision and clarity of black-box metrics such as average dialogue length or glass-box metrics such as word-error rate for speech recognizers. Nevertheless, in the absence of better criteria, they do provide insight into the abilities of different systems and a path for developing better criteria for evaluating dialogue managers.

The Tick-List evaluation we propose does not apply directly to dialogue design toolkits such as Nuance's Dialog Builder and the CSLU toolkit and we do not here carry out such an evaluation. Those toolkits are the means for building applications which one might then evaluate with the Tick-list.

3.2 Trindi Tick-list

The trindi wish list of desired dialogue behaviour is specified as a "tick-list" of yes-no questions. The metric is therefore a qualitative one whose minimal scale is "yes" and "no". We expect the tick-list to develop both in terms of the number and content of the questions themselves and in the range of possible answers. The questions to be ticked or crossed (together with illustrative examples) are listed below.

Qn 1 *Is utterance interpretation sensitive to context?*

(3.1) When do you want to arrive?

(3.2) tomorrow

(3.3) You want to arrive on Friday 3rd January

(3.4) When do you want to arrive ?

(3.5) 6 p m

(3.6) You want to arrive at 6 p m

(3.7) When do you want to leave ?

(3.8) 10 p m

(3.9) You want to leave at 10 p m

There are a number of ways in which interpretation can be context sensitive. First, there are deictic cases (e.g. tomorrow, next week) where interpretation is sensitive to the context of utterance. In order to test for these cases, one needs to attempt to answer questions using deictic expressions (here, now, tomorrow etc.)

Secondly, there are 'discourse cases' where interpretation is sensitive to the interpretation of the dialogue so far. The proper test for examples in the second case is to use an answer (e.g. '6 p m') that could answer other questions in the dialogue than the one currently being posed. '6 p m' could be used to answer either 3.4 or 3.7 in the example above. Consequently the fact that it *is* taken as denoting the arrival time shows that interpretation was suitably sensitive to dialogue context in the form of the question which was just asked.

Qn 2 *Can the system deal with answers to questions that give more information than was requested?*

(3.10) Where do you want to go?

(3.11) Boston at three p.m.

In this example, the question is answered by "Boston" but additional information "at three p.m." is supplied which the responder believes to be helpful in some way. The most straightforward case, and the one we are interested in here, is where the extra information *would* be elicited by a further exchange were the user 'merely' to answer the original question, for example

(3.12) Where do you want to go?

(3.13) Boston

(3.14) When do you want to go?

(3.15) Three p m.

In this way, a dialogue can be compressed, if the dialogue manager is able to update its information state with information 'beyond its own expectations'.

In order to test this dialogue feature, one should first construct a dialogue consisting only of 'mere' answers and then test whether one can shorten the resulting dialogues by compounding answers together.

Qn 3 *Can the system deal with answers to questions that give different information than was actually requested*

(3.16) Where do you want to go?

(3.17) I'm leaving New York on the 29th June.

In this example, the original question is not answered at all but other information is supplied which, again, would be elicited by further exchanges if the user had answered the original question, for example:

(3.18) Where do you want to go?

(3.19) Boston

(3.20) When do you want to go?

(3.21) Three p m

(3.22) Where are you starting from?

(3.23) New York

In this way, the user can re-direct the canonical order of the dialogue. Consequently, in order to test this feature, one should first construct a dialogue consisting of 'mere' answers and then test whether one can re-order the dialogue. (It is possible that the application is such that re-ordering is not feasible).

Qn 4 *Can the system deal with answers to questions that give less information than was actually requested*

(3.24) What date do you want to go?

(3.25) Sometime next week

(3.26) I need to know which day next week

(3.27) Tuesday

In this example, the original question is answered but not sufficiently precisely for the system. Nevertheless, the continuing dialogue shows that the system was able to make use of the information that was supplied. Making use of the information that was supplied is important for distinguising the case of supplying *no* information from that of *some* information.

In the general case, the user is able to find his way to the appropriate level of information specification. Consequently, in order to test this feature, one should first construct dialogues consisting of answers that are already at the expected level of precision and then test whether one can negotiate one's way towards that level using less precise information.

Qn 5 *Can the system deal with ambiguous designators?*

(3.28) Where do you want to go?

(3.29) Berlin

(3.30) Berlin-Ostbahnhof?

(3.31) That's right

This is another example of a question being answered but not sufficiently precisely for the system. We separate this from the previous question in order to highlight the importance of referential ambiguity and distinguish it from cases of vagueness as above.

Qn 6 *Can the system deal with negatively specified information?*

(3.32) When do you want to leave?

(3.33) Not before lunchtime

In order to test this feature, one should first construct a dialogue where questions are answered positively and then construct a test dialogue where the same information is specified negatively. For example,

(3.34) When do you want to leave?

(3.35) After lunch

constitutes a suitably positively formulated dialogue.

Qn 7 *Can the system deal with no answer to a question at all?*

(3.36) Where do you want to go?

(3.37) ...

In this example, no answer is forthcoming to the question at all. The user may have abandoned the dialogue or he may be abstaining from answering the question either because he cannot or will not.

In order to test this feature, one can simply fail to answer a question and see how the system reacts.

Qn 8 *Can the system deal with noisy input?*

(3.38) Where do you want to go?

(3.39) *... noise ...*

In this example, no meaningful answer or other contribution to the dialogue is forthcoming. There may be problems with communication channels or the user may not be dialogue cooperative.

In order to test this feature, one needs to simulate the input of noise to the system.

Qn 9 *Can the system deal with 'help' sub-dialogues initiated by the user?*

(3.40) What meal do you want?

(3.41) What are the choices?

(3.42) Standard, Vegetarian, Kosher and Halal

(3.43) Halal, please

In this example, the question is not answered but the response opens a sub-dialogue whose completion shows how the question can be answered. In general, dialogue systems need to provide help just as much as any other sort of interactive system. The help should be context sensitive.

Qn 10 *Can the system deal with 'non-help' sub-dialogues initiated by the user?*

(3.44) Where do you want to go?

(3.45) Are there any economy flights to Boston today?

(3.46) yes

(3.47) I want to go to Boston today and New York on Thursday

In this example, the question is not answered even though the user understands how to answer it. The answer he wishes to give is dependent on information possessed by the system and which would be elicited if the question were answered.

(3.48) Where do you want to go?

(3.49) Boston today and New York on Thursday

(3.50) Do you want an economy flight today?

(3.51) Yes

(3.52) There are no economy flights to Boston today

(3.53) Ok. I want to go to New York today and Boston on Thursday

In this way, the user is again able to redirect the order of dialogue processing to suit his purposes.

Qn 11 *Does the system only ask appropriate follow-up questions?*

(3.54) How would you like to travel?

(3.55) By train

(3.56) What class?

(3.57) First

(3.58) When do you want to travel?

(3.59) How would you like to travel?

(3.60) By car

(3.61) When do you want to travel?

This feature tests whether the system's own pattern of questioning is sensitive to the answers that it has already received. In the example shown, it only makes sense to inquire for the class of travel if travel is by train and not by car.

Qn 12 *Can the system deal with inconsistent information?*

(3.62) I want to fly tomorrow on Tuesday.

(3.63) Tomorrow is Wednesday

This feature tests whether or how much inference the system can perform over its own information states.

3.3 Evaluations from the theoretical perspective

3.3.1 The Philips System

Qn 1 *Is utterance interpretation sensitive to context?*

It is an interesting feature of the Philips System that utterance interpretation appears (according to the published descriptions) *not* to be dialogue sensitive. As indicated earlier (2.2.1), the system appears not to maintain a record of the question it has just asked. Consequently, in order to be interpretable, answers must generally contain sufficient linguistic cues within them in order to be correctly interpreted. As noted earlier, there do appear to be hooks to permit certain context-sensitive behaviours to be added to the system.

One instance of context sensitivity in the Philips System is documented (there may be others, undocumented or documented elsewhere). The meanings of certain date expressions e.g. 'tomorrow', 'next Saturday' are defined with respect to the current date. So, 'tomorrow' is represented as 'Current date + 1' and 'next Saturday' is defined as 'the first day after the Current date whose day-within-week number is 6'.

Qn 2 *Can the system deal with answers to questions that give more information than was requested?*

An interesting result of the above mentioned dialogue *insensitivity* is that it is straightforward for the Philips System to handle more information than requested. Since the utterance interpretation process is not being primed to expect certain sorts of information, there is no danger of such priming leading to ignoring *other* useful information.

Qn 3 *Can the system deal with answers to questions that give different information than was actually requested?*

Similarly, since user inputs are expected to contain sufficient linguistic cues for their own interpretation and the system is not being primed to expect only certain sorts of information, the Philips system will handle whatever information is provided in the same manner whenever it is provided.

Qn 4 *Can the system deal with answers to questions that give less information than was actually requested?*

With each expected slot to fill, the Philips System can associate a set of disambiguation questions. One of the conditions designed for such questions is 'insufficiently specified values'.

Qn 5 *Can the system deal with ambiguous designators?*

In the Philips System, it appears that ambiguous designators are dealt with as an instance of insufficiently specified values.

Qn 6 *Can the system deal with negatively specified information*

Descriptions of the Philips System include verification dialogues which include negatively specified information such as 'No, Homburg' cited above. Otherwise, no special mention is made of negatively specified information.

Qn 7 *Can the system deal with no answer to a question at all?*

There is no mention of the system's strategy in this regard.

Qn 8 *Can the system deal with noisy input?*

There is no mention of the system's strategy in this regard.

Qn 9 *Can the system deal with 'help' sub-dialogues initiated by the user?*

There is no mention of such a facility.

Qn 10 . *Can the system deal with 'non-help' sub-dialogues initiated by the user?*

There is no mention of such a facility.

Qn 11 . *Does the system only ask appropriate follow-up questions?*

The system architecture is designed in order to facilitate dialogues which do only ask appropriate follow-up questions.

Qn 12 . *Can the system deal with inconsistent information?*

With each expected slot to fill, the Philips System can associate a set of disambiguation questions. One of the conditions designed for such questions is precisely the presence of inconsistent information, for example, the presence of "next Thursday" and "March 3rd" when next Thursday is not the 3rd of March.

3.3.2 The Autoroute System

Qn 1 *Is utterance interpretation sensitive to context?*

The autoroute system uses SRI's Core Language Engine as its chief linguistic processor, a machine for processing language in context, where context includes both linguistic and non-linguistic features.

Qn 2 *Can the system deal with answers to questions that give more information than was requested?*

As noted in the earlier discussion of Autoroute (2.2.2), the system does configure itself to expect an answer to a question. Consequently, extra information can be provided because the notion of answer implemented in the system does not exclude the provision of extra information. (Given a notion of something strictly *only* answering a question, an autoroute answer is anything that subsumes such a strict answer).

Qn 3 *Can the system deal with answers to questions that give different information than was actually requested?*

In principle, this is possible within the autoroute system. The theory of conversational games which the system implements includes the possibility of the system abandoning the game that it thought it was playing (abandoning the question it asked) and instead adopting the new one proposed by the user. Configuring the system to produce such behaviour appears to be a non-trivial task.

Qn 4 *Can the system deal with answers to questions that give less information than was actually requested?*

There is no mention of such a facility.

Qn 5 *Can the system deal with ambiguous designators?*

There is no mention of such a facility within the dialogue manager. The Core Language Engine maintains a history list of objects referred to in order to aid with disambiguation.

Qn 6 *Can the system deal with negatively specified information*

Descriptions of the autoroute include verification dialogues which include negatively specified information such as 'No, Bristol'. Otherwise, no special mention is made of negatively specified information.

Qn 7 *Can the system deal with no answer to a question at all?*

There is no mention of the system's strategy in this regard.

Qn 8 *Can the system deal with noisy input?*

There is no mention of the system's strategy in this regard.

Qn 9 *Can the system deal with 'help' sub-dialogues initiated by the user?*

There is no mention of such a facility. In principle, user-initiated sub-dialogues are a possibility within conversational game theory and its autoroute implementation.

Qn 10 . *Can the system deal with 'non-help' sub-dialogues initiated by the user?*

There is no mention of such a facility. In principle, user-initiated sub-dialogues are a possibility within conversational game theory and its autoroute implementation.

Qn 11 . *Does the system only ask appropriate follow-up questions?*

The system architecture is designed in order to facilitate dialogues which do only ask appropriate follow-up questions.

Qn 12 . *Can the system deal with inconsistent information?*

There is no mention of such a facility.

3.3.3 The Trains System

Qn 1 *Is utterance interpretation sensitive to context?*

The Trains linguistic processor is designed to be context-sensitive in several ways.

Qn 2 *Can the system deal with answers to questions that give more information than was requested?*

Trains is not primarily a question-answering system. However, it certainly is designed to permit analyses of user-inputs in terms of multiple speech acts. For example, [Allen *et al.*(1996a)] cites the example "Okay now let's take the last train and go from Albany to Milwaukee" which is analysed as a sequence of three speech acts: an acknowledgment, a telling, and a request. In principle, the current goal of the dialogue manager (in fact, there is a stack) may be met by one of these acts but the others will still, if possible, be processed. Consequently, at least with respect to this level of granularity of speech acts, extra information can be accommodated.

Qn 3 *Can the system deal with answers to questions that give different information than was actually requested?*

Again, Trains is not primarily a question-answering system. However, the system is designed to attempt to infer the user's overall goals at every stage and act accordingly. It is possible in principle to make the system attach a higher weight to adopting a goal inferred from the latest utterance than to rejecting its interpretation of the current utterance based on what the system believes was the current goal.

Qn 4 *Can the system deal with answers to questions that give less information than was actually requested?*

Again, Trains is not primarily a question-answering system. In general, it ought to be possible for the system to successfully deal with such information. Since the system is continually building and refining a plan, it certainly ought to be possible to plan at different levels of abstraction.

Qn 5 *Can the system deal with ambiguous designators?*

Trains is built with an explicit discourse referent stack in order to help correctly determine the reference of any ambiguous designators.

Qn 6 *Can the system deal with negatively specified information*

There is no mention of the system's strategy in this regard.

Qn 7 *Can the system deal with no answer to a question at all?*

There is no mention of the system's strategy in this regard.

Qn 8 *Can the system deal with noisy input?*

The version of Trains which is the subject of this document has several components explicitly designed to help copy with noisy input.

Qn 9 *Can the system deal with 'help' sub-dialogues initiated by the user?*

There is no mention of such a facility.

Qn 10 . *Can the system deal with 'non-help' sub-dialogues initiated by the user?*

In general, Trains dialogues are designed with user initiative in mind.

Qn 11 . *Does the system only ask appropriate follow-up questions?*

In general, the system is not designed for asking questions.

Qn 12 . *Can the system deal with inconsistent information?*

There are facilities within the system for detecting and coping with various sorts of inconsistency.

3.4 Empirical test results

3.4.1 The Philips System -German

The following tests were carried out on Fahrplanauskunft (Philips Aachen) on telephone number +(0)241-604020 in September 1998 by Johan Bos and Manfred Pinkal. Each test was performed twice or three times.

Qn 1 *Is utterance interpretation sensitive to context?*

Yes, in cases where names of days as given by the user are rendered into deictic expressions, such as "tomorrow". Asking on a thursday:

User: "Ich möchte am Freitag fahren."
(*I would like to travel on Friday.*)

System: "Sie möchten also morgen fahren."
(*So, you would like to travel tomorrow.*)

No, for resolution of time designators and names. Witnessing the following example dialogue:

User: "Ich möchte am 5 Uhr ankommen."
(*I would like to arrive at 5 am.*)

System: "Sie möchten also um 5 Uhr morgens ankommen."
(*So you would like to arrive at 5 am.*)

It would be more likely to understand "arriving at 5 o'clock" as arriving at 5 pm, rather as the system suggests, 5 am.

Qn 2 *Can the system deal with answers to questions that give more information than was requested?*

Yes.

System: "Von wo nach wo möchten Sie fahren?"
(*From where to where do you want to go?*)

User: "Ich möchte gerne am Montag nach Berlin fahren."
(*I would like to go to Berlin on Monday.*)

System: "Von wo möchten Sie am Montag den 21. September nach Berlin fahren?"
(*From where would you like to go to Berlin on Monday September 21st?*)

The Philips system always opens the dialogue by asking for place of departure and destination; user answers with destination and time. The system's next question concerns the departure place only, which one would expect.

Qn 3 *Can the system deal with answers to questions that give different information than was actually requested?*

Yes. In cases where user responds to something not explicitly asked for by the system:

System: "Von wo nach wo möchten Sie fahren?"
(*From where to where do you want to go?*)

User: "Ich möchte gerne am Montag fahren."
(*I would like to go on Monday.*)

System: "Von wo nach wo möchten Sie am Montag den 21. September fahren?"
(*From where to where would you like to go on Monday September 21st?*)

Here the system asks for place of departure and destination. User responds with the date of travel — the system accepts this.

Qn 4 *Can the system deal with answers to questions that give less information than was actually requested?*

Yes.

System: "Von wo nach wo möchten Sie fahren?"
(*From where to where do you want to go?*)

User: "Ich möchte gerne nach Amsterdam fahren."
(*I would like to go to Amsterdam.*)

System: "Von wo möchten Sie nach nach Amsterdam fahren?"
(*From where do you want to go to Amsterdam?*)

This is also true for more difficult cases as:

System: "Wann wollen Sie fahren?"
(*When do you want to go?*

User: "Nächste Woche."
(*Next week, please.*)

System: "Am welchen Tag genau wollen Sie fahren."
(*On exactly which day do you want to go?*)

User responds to the system's question with a general answer; systems asks for a more specific one.

Qn 5 *Can the system deal with ambiguous designators?*

This test applies to names of places. The Philips systems always undertakes default resolution strategy, both in preferences ("Berlin" is rendered as "Berlin-Ostbahnhof"), and in equally likely candidates ("Neustadt" is rendered as "Neustadt-Weinstrasse", although this is only one of the many Neustadts in Germany).

System ignores vague modifiers in users answers:

System: "Wann wollen Sie am Montag fahren?"
(*When would you like to go on Monday?*)

User: "Irgendwann am späten Vormittag."
(*Somewhere in the late morning.*)

System: "Sie möchten also zwischen 8 und 12 Uhr fahren."
(*So, you would like to go between 8 and 12?*)

Qn 6 *Can the system deal with negatively specified information?*

Sometimes yes, sometimes no. Here is an example where negation was not recognized:

User: "Ich möchte nicht for Freitag fahren."
(*I wouldn't like to go before Friday.*)

System: "Sie möchten also am Freitag fahren."
(*So you would like to go on Friday.*)

And here one where negation is recognized.

System: "Von wo nach wo möchten Sie fahren?"
(*From where to where do you want to go?*)

User: "Ich möchte von Saarbrücken nach Frankfurt fahren."
(*I would like to go to from Saarbrücken to Frankfurt.*)

System: "Sie wollen also von Saarbrücken nach Frankfurt Main fahren."
(*So you want to go from Saarbrücken to Frankfurt Main.*)

User: "Nee, ich möchte NICHT nach Frankfurt fahren."
(*No, I do NOT want to go to Frankfurt.*)

System: "Wohin wollen Sie fahren?"
(*Where do you want to go?*)

Qn 7 *Can the system deal with no answer to a question at all?*

If the system does not recognize or if the user waits too long the system reponds with a "I'm sorry" and repeats the previous question.

Qn 8 *Can the system deal with noisy input?*

It deals with it by making a guess. Here is a dialogue where the user responds with some non-linguistic sounds.

System: "Von wo nach wo möchten Sie fahren?"
(*From where to where do you want to go?*)

User: "Ich möchte gerne nach Amsterdam fahren."
(*I would like to go to Amsterdam.*)

System: "Von wo möchten Sie nach nach Amsterdam fahren?"
(*From where do you want to go to Amsterdam?*)

User: "xxx xxxx xxxx xxxx." (some verbal noise)

System: "Sie möchten also von Heidelberg nach Amsterdam fahren."
(*So you want to go from Heidelberg to Amsterdam.*)

Here is an example dialogue where the user responds, while at the same time there is a lot of background noise. System makes the wrong guess.

System: "Von wo nach wo möchten Sie fahren?"
(*From where to where do you want to go?*)

User: "Ich möchte gerne nach Amsterdam fahren."
(*I would like to go to Amsterdam.*)

System: "Von wo möchten Sie nach nach Amsterdam fahren?"
(*From where do you want to go to Amsterdam?*)

User: "Von Saarbrücken." (with background noise)
(*From Saarbrücken.*) (with background noise)

System: "Sie möchten also von Stuttgart nach Amsterdam fahren."
(*So you want to go from Stuttgart to Amsterdam.*)

Qn 9 *Can the system deal with 'help' sub-dialogues initiated by the user?*

No. An attempt on a more general answer results in a response saying that the user wasn't understood. Then the initial question is repeated. Here is an example:

System: "Von wo nach wo möchten Sie fahren?"
(*From where to where do you want to go?*)

User: "Ich hätte gerne eine Fahrplanauskunft."
(*I would like to have a route, please.*)

System: "Ich habe Sie nicht verstanden."
(*Sorry, I didn't understand you.*)

Sub-dialogues such as asking whether a specific day is a holiday are not processed.

Qn 10 . *Can the system deal with 'non-help' sub-dialogues initiated by the user?*

Yes, if it fits in the domain. Such as (relative wording underlined):

System: "Wann wollen Sie fahren?"
(*When would you like to travel?*)

User: "Ich möchte gerne um 18 Uhr ankommen."
(*I would like to arrive at 6 pm.*)

System: "Sie möchten also um 18 Uhr ankommen."
(*So, you want to arrive at 6 pm.*)

Systems ask when the user wants to travel. User responds with destination time.

Qn 11 . *Does the system only ask appropriate follow-up questions?*

Not tested.

Qn 12 . *Can the system deal with inconsistent information?*

Proposing different values for one slot:

User: "Ich möchte am Montag fahren."
(*I would like to go on Monday.*)

System: "Sie möchten also am Montag fahren."
(*So you want to go on Monday.*)

User: "Ich möchte am Dienstag fahren."
(*I would like to go on Tuesday.*)

System: "Sie möchten also am Dienstag fahren."
(*So you want to go on Tuesday.*)

System replaces monday by tuesday, and leaves inconsistency unnoticed. Another test:

User: "Ich möchte von Berlin nach Berlin fahren."
(*I would like to go from Berlin to Berlin.*)

System doesn't understand the user, and repeats the question.

3.4.2 The Philips System - Swedish

The following tests were carried out on the public SJ (swedish railway) telephone service on telephone number +(0)20-757575 in February 1999 by Staffan Larsson and Peter Bohlin.

Qn 1 *Is utterance interpretation sensitive to context?*

- deictic: Yes.

  System: när vill du resa?
  (when do you want to travel?)
  User: i morgon
  (tomorrow)
  System: i morgon fredagen den femte februari, stämmer det?
  (tomorrow, the fifth of february, is that correct?)

- local discourse: Yes.

  System: varifrån och vart vill du resa?
  (from where and to where do you want to travel)
  User: göteborg

  System: från göteborg central, stämmer det?
  (from göteborg central, is that correct?)
  User: ja
  (yes.)

System: vart vill du resa?
(where do you want to go?)

User: stockholm

System: till stockholm central, stämmer det?
(to stockholm central, is that correct?)

Qn 2 *Can the system deal with answers to questions that give more information than was requested?*

No.

System: varifrån och vart vill du resa?
(from where and to where do you want to travel)

User: från göteborg till stockholm den femte februari
(from göteborg to stockholm on the fifth of february)

System: från göteborg central till stockholm central, stämmer det?
(from göteborg central to stockholm central, is that correct?)

User: ja
(yes.)

System: när vill du resa?
(when do you want to travel?)

Qn 3 *Can the system deal with answers to questions that give different information than was actually requested?*

No.

System: varifrån och vart vill du resa?
(from where and to where do you want to travel)

User: jag vill resa den femte februari
(i want to travel on the fifth of february)

System: från lessebo, stämmer det?
(from lessebo, is that correct?) ....

System: när vill du resa?
(when do you want to travel?)

Qn 4 *Can the system deal with answers to questions that give less information than was actually requested?*

- Answer to part of complex question: Yes.

  System: varifrån och vart vill du resa?
  (from where and to where do you want to travel)

  User: från göteborg ...
  (from göteborg)

  System: vart vill du resa?
  (where do you want to go?)

- Partial answer to single question: Yes.

  System: när vill du resa?
  (when do you want to travel?)

  User: nästa vecka
  (next week)

  System: vilken dag under perioden vill du resa?
  (which day during the period do you want to travel?)

Qn 5 *Can the system deal with ambiguous designators?*

Yes.

System: varifrån och vart vill du resa?
(from where and to where do you want to travel)

User: från stockholm
(from stockholm)

System: från stockholm central, stämmer det?
(from stockholm central, is that correct?)

(There is also a station called "stockholm södra")

Qn 6 *Can the system deal with negatively specified information?*

No.

System: när vill du resa?
(when do you want to travel?)

User: inte före klockan ett
(not before one o'clock)

System: före klockan tretton, stämmer det?
(before one o'clock, is that correct?)

Qn 7 *Can the system deal with no answer to a question at all?*

Yes. (but see comment)

System: varifrån och vart vill du resa?
(from where and to where do you want to travel)

User: (silence)

System: säg vart du vill resa
(tell me where you want to go)

User: (silence)

System: jag förstod inte. säg varifrån och vart du vill resa.
(i didn't understand. tell me from where and to where you want to travel.)

Qn 8 *Can the system deal with noisy input?*

No.

System: varifrån och vart vill du resa?
(from where and to where do you want to travel)

User: (makes noise by blowing air into telephone mic)

System: till örnsköldsvik från nässjö central, stämmer det?
(to örnsköldsvik from nässjö central, is that correct?)

Qn 9 *Can the system deal with 'help' sub-dialogues initiated by the user?*

No.

System: ungefär vilken tid vill du resa?
(approximately when do you want to travel?)

User: går det några tåg före klockan nio?
(are there any trains leaving before nine o'clock?)

System: före klockan nio, stämmer det?
(at nine o'clock, is that correct?)

Comment: the system does not recognize the question but treats it as an answer: "before nine o'clock"

System: varifrån och vart vill du resa?
(from where and to where do you want to travel)

User: vilken station ligger närmast örkelljunga?
(what station is closest to örkelljunga?)

System: till ljungaverk, stämmer det?
(to ljungaverk, is that correct?)

Qn 10 *Can the system deal with 'non-help' sub-dialogues initiated by the user?*

No.

System: vart vill du resa?
(to where do you want to travel)

User: finns det några x2000-tåg till malmö?
(are there any x2000 trains to malmö?)

System: är det rätt uppfattat att du vill åka till malmö central?
(is it correctly understood that you want to go to malmö central?)

System: vart vill du resa?
(to where do you want to travel)

User: till stockholm på måndag och till lund på torsdag
(to stockholm on monday and lund on thursday)

System: är det rätt uppfattat att du vill åka till stockholm central?
(is it correctly understood that you want to go to stockholm central?) ... (no further utterances relating to lund)

Qn 11 *Does the system only ask appropriate follow-up questions?*

Not tested.

Qn 12 *Can the system deal with inconsistent information?*

Partially

Note: this dialogue took place on a thursday.

User: i morgon tisdag
(tomorrow tuesday)

System: tisdagen den sextonde februari, stämmer det?
(tuesday the sixteenth of february, is that correct?)

User: nä
(no)

System: imorgon fredag, stämmer det?
(tomorrow friday, is that correct?)

3.4.3 The SRI Autoroute System

SRI International's Autoroute Demo for Trindi, available via

`http://www-cgi.cam.sri.com/trindi`

Tested on Oct 14, 1998 (Johan Bos). There are two versions of the Trindi demo to choose from. In the basic demo, the system asks you questions and you simply supply the answers. The system's strategy in negotiating the dialogue is fixed. In the advanced demo, you also take over responsibility for part of the system's strategy.

Qn 1 *Is utterance interpretation sensitive to context?*

Asserting that the user wants to travel in the afternoon makes the system asks "Is the departure time fourteen hundred hour ?".

Answering this question with "no, at six" would be appropriate to test context sensitivity (the system should infer that 6 p.m. was meant, not 6 a.m.). However, Autoroute refuses to recognize phrases like "six" in this stage of the dialogue.

Qn 2 *Can the system deal with answers to questions that give more information than was requested?*

Yes.

System: "Where do you want to go?"

User: "From Edinburgh to Glasgow"

System: "Is the origin Edinburgh (the place)?"

Autoroute (always) opens the dialogue by asking for the destination only. user answers with destination and time. The system's next move is to ask for confirmation for both origin and destination.

Qn 3 *Can the system deal with answers to questions that give different information than was actually requested?*

Not tested.

Qn 4 *Can the system deal with answers to questions that give less information than was actually requested?*

The system doesn't know constructions like "around eight a m". The system doesn't analyse phrases such as "next week".

Yes, if it recognizes the input. As in:

System: "Where do you want to go?"

User: "From Edinburgh at two p m"

System: "Please state only the name of your destination."

The information on origin and time is not recorded, because the systems asks for origin and time later in the dialogue.

Qn 5 *Can the system deal with ambiguous designators?*

This test applies to names of places. Tried with "Edwinstowe", and "Newcastle", but these are handled as being unique designators.

"Great Malvern" and "Malvern" are both analysed as "Great Malvern" (you can tell from the final route description). However, asking again for confirmation as in the following dialogue seems superfluous:

System: "Where do you want to go?"

User: "Malvern"

System: "Is the destination Malvern (the place) ?"

User: "Great Malvern"

System: "Is the destination Great Malvern (the place) ?"

Qn 6 *Can the system deal with negatively specified information?*

The user is able to deny a particular instance of a slot, if the system asks for confirmation on a particular value. It doesn't analyze constructions like "not before eight a m"

Qn 7 *Can the system deal with no answer to a question at all?*

If the user waits too long the system reponds with a "What" and repeats the previous question. It repeats this as many times till it receives an appropriate answer.

Qn 8 *Can the system deal with noisy input?*

If the system does not recognize the input the system responds with a "What" and repeats the previous question. It repeats this as many times till it receives an appropriate answer.

Qn 9 *Can the system deal with 'help' sub-dialogues initiated by the user?*

Not tested.

Qn 10 . *Can the system deal with 'non-help' sub-dialogues initiated by the user?*

Not tested.

Qn 11 . *Does the system only ask appropriate follow-up questions?*

Another interesting example, where a question is responded with an inappropriate answer:

System: "Do you want the quickest route or the shortest route?"

User: "yes"

At this response the system repeats the question, and the user answers the choice-question with "no".

System: "Do you want the quickest route or the shortest route?"

User: "no"

The system then quits the session, which is, from a logical point of view, correct (disjunction narrow scope reading).

Qn 12 . *Can the system deal with inconsistent information?*

Proposing different values for one slot is possible at a later stage of the dialogue if the actual questions is answered as well:

System: "Where do you want to go?"

User: "Edinburgh"

System: "Is the destination Edinburgh (the place) ? "

User: "yes"

System: "Where does your route start?"

User: "The destination is London"

System: "Where does your route start?"

User: "I want to go from Edinburgh to London"

System: "Is the origin Edinburgh (the place) ? "

User: "yes"

System: "Is the destination London (the place) ? "

Another test shows that the system is not sensitive to queries that make no sense:

System: "Where do you want to go?"

User: "Edinburgh"

System: "Is the destination Edinburgh (the place) ?"

User: "yes"

System: "Where does your route start?"

User: "Edinburgh"

System: Is the origin Edinburgh (the place) ?"

Neither does the system recognizes an inconsistent dialogue that is constructed by first asserting a particular value for a slot, then denying it, and after that proposing the same value again.

3.4.4 The Trains System

Qn 1 *Is utterance interpretation sensitive to context?*

Yes.

User: "Move the Boston train to Albany."

System: "Ok."

Various types of anaphoric expression, including pronouns and definite descriptions (as above) are interpreted in context.

Qn 2 *Can the system deal with answers to questions that give more information than was requested?*

Unclear. TRAINS almost never asks questions.

Qn 3 *Can the system deal with answers to questions that give different information than was actually requested?*

Unclear. Again, TRAINS never really takes the initiative in this way.

Qn 4 *Can the system deal with answers to questions that give less information than was actually requested?*

Unclear. Same problem as with 2 and 3.

Qn 5 *Can the system deal with ambiguous designators?*

Yes, assuming that definite expressions and suchlike are 'ambiguous designators':

User: "Move the train back."

System: "Yes."

Here 'the train' is arguably ambiguous in that there are many objects fitting that description; TRAINS interprets the expression correctly based on the context.

Qn 6 *Can the system deal with negatively specified information?*

No. TRAINS does not seem to accept obvious kinds of negative information:

User: "Do not move the Richmond train."

System: "Sorry. I don't understand your destination for the engine at Richmond; if I haven't mangled your meaning beyond all comprehension."

The parser does not handle the 'not' in the user's input at all. In the following dialogue, there are 3 trains in the context, so the system's response is peculiar.

User: "Move every train to Scranton except the New York train."

System: "Ok. Unfortunately, I can't seem to find a train."

Qn 7 *Can the system deal with no answer to a question at all?*

Yes, in a sense. The only kind of questions the system asks are things like "are you there?" if it does not get input for a while. If there is no response, the system will eventually close down.

Qn 8 *Can the system deal with noisy input?*

Yes. It was not possible to test this in Edinburgh, but the example runs of the system indicate that it can deal with many speech processing problems, including noise.

Qn 9 *Can the system deal with 'help' sub-dialogues initiated by the user?*

Unclear. It's not obvious what a 'help' dialogue is in the context of TRAINS. See the answer to 10.

Qn 10 . *Can the system deal with 'non-help' sub-dialogues initiated by the user?*

Yes. The user can ask various kinds of question, which are interpreted with different degrees of success:

User: "Can we move the New York train to Scranton?"

System: "Ok."

User: "Where is the Milwaukee train?"

System: "The engine at Milwaukee is at Milwaukee."

User: "How far is it from New York to Scranton?"

System: "The current route from New York to Scranton is 151 miles long."

Qn 11 . *Does the system only ask appropriate follow-up questions?*

No. Again, the system never really asks questions.

Qn 12 . *Can the system deal with inconsistent information?*

No. Not always, anyway - in the following interchange there is no train at Columbus, and there never has been throughout the dialogue:

User: "Move the Columbus train to Toledo."

System: "Ok."

3.5 Conclusions

In table 3.1, we summarize the results of our evaluations. Each system has two columns *Theory* and *Emp.* corresponding to the theoretical investigation and the empirical investigation respectively. For the Philips system, we list only the results found for the German train timetable system. In general, the Swedish version of the system seemed to perform less well, which we hypothesize may be due to poorer performance of the speech recognition component (see also the comments below).

	Philips		*Autoroute*		*Trains*	
Qn	*Theory*	*Emp.*	*Theory*	*Emp.*	*Theory*	*Emp*
Q1	Yes (*peri.*)	Partially	Yes	Partially	Yes	Yes
Q2	Yes	Yes	Yes	Yes	Yes (*peri.*)	Unclear
Q3	Yes	Yes	Yes (*peri.*)	not tested	Yes (*peri.*)	Unclear
Q4	Yes	Yes	No mention	Yes	Yes (*peri.*)	Unclear
Q5	Yes	Yes	Yes	No	Yes	Yes
Q6	Partially	Partially	Partially	Partially	No mention	No
Q7	No mention	Partially	No mention	Partially	No mention	Yes
Q8	No mention	Yes	No mention	Yes	Yes	Yes
Q9	No mention	No	No mention	not tested	No mention	Unclear
Q10	No mention	Yes	No mention	No	Yes	Yes
Q11	Yes	not tested	Yes	No	n/a	No
Q12	Yes	No	No mention	No	Yes	No

Table 3.1: Summary Table of Results

It is important not to over-interpret the contents of table 3.1. First, as we have stressed elsewhere in this document, the systems and components examined strongly reflect the nature of the research and commercial projects within which they have been created and those projects vary very greatly. Direct cross-system comparisons may not be meaningful. Secondly, it is not always easy to be sure that behaviour manifested by a system is a property of the dialogue manager itself (or some other component, or some property of the environment and the system together) and, even if it is, whether the instance found is an instance of a general behavioural capability, rather than a one-off or partially systematic reflex. In the theoretical evaluations, a column is marked as "Yes (*peri.*)" meaning "Yes, peripherally" if the behaviour can occur but does not appear to be a central feature of the dialogue management model.

It may be surprising at first that the contents of the *Theory* and *Emp.* columns do not match up precisely and, ideally, one would of course expect the results of the two examinations to be the same. In practice, this may not be so for several reasons. First, the behaviour of the tested system may not be that of the described system because the two systems are not the same. 'Advanced' dialogue behaviours which are interesting to publish accounts of may never find their way into final publicly available versions of the system. Also, research prototypes are under constant development so that the latest versions may no longer manifest, or not manifest so easily or obviously, behaviours ascribed to earlier versions. Secondly, it may prove difficult empirically to verify instances of behaviour described in published accounts. For instance, dialogue management behaviour is generally highly dependent on

the behaviour of other components (e.g. the speech recognizer and the parser). If those components perform sufficiently poorly during an empirical test (for example, because the speech recognizer performs particularly poorly with respect to the particular accent or dialect of the evaluator), then features of dialogue management behaviour may never be brought to light. Similarly, published accounts of dialogue managers are naturally written by system experts who understand the environment and possible behaviours of system components very well. Behaviours are described from this perspective. An evaluator who is not so familiar with the intended environment or system components simply may not be able to find ways to reproduce instances of the claimed behaviour. Finally, it is also a possibility that the published accounts may simply misrepresent the system.

Bibliography

[Alexandersson *et al.*(1997)] Alexandersson, J., Reithinger, N., and Maier, E. (1997). Insights into the Dialogue Processing of VERBMOBIL. Verbmobil-Report 191, DFKI GmbH Saarbrcken.

[Alexandersson *et al.*(1998)] Alexandersson, J., Buschbeck-Wolf, B., Fujinami, T., Kipp, M., Maier, E., Reithinger, N., Schmitz, B., and Siegel, M. (1998). Dialogue Acts in VERBMOBIL-2 – Second Edition. Verbmobil-Report 226, DFKI GmbH Saarbrcken.

[Allen *et al.*(1996a)] Allen, J., Miller, B., Ringger, E., and Sikorski, T. (1996a). A robust system for natural spoken dialogue. In *Proceedings of 34th ACL Santa Cruz*, pages 62–70.

[Allen *et al.*(1994)] Allen, J. F., Schubert, L. K., Ferguson, G., Heeman, P., Hwang, C. H., Kato, T., Light, M., Martin, N. G., W, M. B., Poesio, M., and Traum, D. R. (1994). *The TRAINS Project: A case study in building a conversational planning agent.* University of Rochester, Rochester, New York.

[Allen *et al.*(1996b)] Allen, J. F., Miller, B. W., Ringger, E. K., and Sikorski, T. (1996b). *Robust Understanding in a Dialogue System.* ACL 34.

[Alshawi(1992)] Alshawi, H. (1992). *The Core Language Engine.* M.I.T.Press.

[Aust and Oerder(1995)] Aust, H. and Oerder, M. (1995). Dialogue control in automatic inquiry systems. In *Proceedings of ESCA Workshop on Spoken Dialogue Systems, Vigso, Denmark*, pages 121–124.

[Aust *et al.*(1995)] Aust, H., Oerder, M., Siede, F., and Steinbiss, V. (1995). A spoken lnaguage enquiry system for automatic train timetable information. *Philips Journal of Research*, **49**(4), 399–418.

[Bub and Schwinn(1996)] Bub, T. and Schwinn, J. (1996). Verbmobil: the evolution of a complex large speech-to-speech translation system. In *Proceedings of ICSLP-96*, pages 2371–2374, Philadelphia PA.

[Bub *et al.*(1997)] Bub, T., Wahlster, W., and Waibel, A. (1997). Verbmobil: The Combination of Deep and Shallow Processing for Spontaneous Speech Translation. In *Proceedings of the 22nd International Conference on Acoustics, Speech, and Signal Processing (ICASSP'97)*, Munich, Germany.

[Davies(1997)] Davies, S. (1997). An investigation into the feasibility of constructing a dialogue system for use in the teaching of english as a foreign language, using the cslu rapid prototyper. M.Sc. Dissertation, University of Edinburgh.

[Ferguson *et al.*(1996)] Ferguson, G., Allen, J. F., Miller, B. W., and Ringger, E. K. (1996). *The Design and Implementation of the TRAINS-96 System: A Prototype Mixed-Initiative Planning Assistant.* University of Rochester, Rochester, New York.

[Gibbon *et al.*(1997)] Gibbon, D., Moore, R., and R., W. (1997). *Handbook of Standards and Resources for Spoken Language Systems.* Mouton de Gruyter, Berlin & New York.

[Grosz and Sidner(1986)] Grosz, B. and Sidner, C. (1986). Attention, intentions and the structure of discourse. *Computational Linguistics*, **12**, 175–204.

[Lewin and Pulman(1995)] Lewin, I. and Pulman, S. (1995). Inference in the resolution of ellipsis. In *Proceedings of ESCA Workshop on Spoken Dialogue Systems, Vigso, Denmark*, pages 53–56.

[Russell *et al.*(1990)] Russell, M., Ponting, K., Peeling, S., Browning, S., Bridle, J., and Moore, R. (1990). The arm continuous speech recognition system. In *Proceedings of ICASSP'90 Albuquerque, New Mexico.*